REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1/31/97	Final

4. TITLE AND SUBTITLE

Siberian Conference on System Informatics

5. FUNDING NUMBERS

G N000149610838

6. AUTHOR(S)

Prof. Larry Wittie

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

The Research Foundation of
State University of New York
Computer Science, SUNY Stony Brook
Stony Brook, NY 11794-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

431-0537A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
Math, Computer & Information Sciences Div.
Code 311/Dr. Andre M. van Tilborg, Director
800 North Quincy St.
Arlington, VA 22217-5660

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

19970213 071

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This grant covered two trips to Russia in summer 1996 connected with the Andrei Ershov Second International Memorial Conference entitled Perspectives of System Informatics (PSI '96), held Tuesday through Friday 25-28 June 1996 in Academgorodok near Novosibirsk, Siberia. It was organized by the Institute of Informatics Systems and the Computing Center of the Siberian Branch of the Russian Academy of Sciences.

This Siberian conference had an unusually wide appeal. There were 69 registered attendees, including 36 from 20 countries outside Russia in Europe, the Americas, Asia, and Australia. Russians came from 7 far-flung cities. Conference presentations were a mix of invited talks plus submitted papers. The keynote speaker was Prof. Niklaus Wirth of Zurich.

There were strong sessions on artificial intelligence, program transformations, and object-oriented programming. Many other topics in parallel computing and systems programming were covered. The program transformation papers represented the work of the key practitioners in the world in an increasingly important field. This conference cast its net over large areas of computer science of interest to Prof. Ershov, founder of the sponsoring Institute of Informatics Systems. Despite its breadth, it was a good conference, full of intelligent, entertaining people.

14. SUBJECT TERMS

Computer Science Research Conference in Akademgorodok, Novosibirsk, Siberia, Russia 1996, A.P. Ershov, Artificial Intelligence, Program Transformations, Parallel Computing

15. NUMBER OF PAGES

9

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	

Siberian Conference on System Informatics

Final report on grant ONR N000149610838 SUNY 431-0537A
Larry Wittie lw@cs.sunysb.edu 4/12/96-9/30/96 $36,090

The Principal Investigator made two trips to Novosibirsk and Moscow Russia in connection with this grant. The grant provided funds to help support the Andrei Ershov Second International Memorial Conference entitled Perspectives of System Informatics (PSI '96), held Tuesday through Friday, 25-28 June 1996. It took place in the House of Scientists conference center, in Academgorodok slightly more than one hour by car from Novosibirsk, Siberia. It was organized by personnel of the Institute of Informatics Systems and the Computing Center, both research organizations in the Siberian Branch of the Russian Academy of Sciences.

1. PSI '96 IN NOVOSIBIRSK JUNE 1996

The Investigator spent four days from 7 AM Tuesday 25 June 1996 to 9 AM Saturday 29 June 1996 near Novosibirsk. This Siberian conference had an unusually wide appeal. There were 69 registered attendees, including 36 from 20 countries outside Russia. Russians came from 7 far-flung cities: Novosibirsk (18), Moscow (6), Pereslavl-Zalessky (4), St. Petersburg (2), Tver, Krasnoyarsk, and Vladivostok. Most (27) foreign attendees were Europeans, from: Germany (7), Denmark (4), Greece (3), Austria (2), Netherlands (2), Yugoslavia (2), Bulgaria, France, Romania, Poland, United Kingdom, Sweden, Switzerland. The other (7) nations represented were: United States (3), Argentina, Australia, China, India, Japan, and Macau.

The conference presentations were a mix of invited talks plus submitted papers. Invited speakers came from Switzerland, Germany, Netherlands, Poland, America (two Russian émigrés), and Russia (Moscow). The Monday afternoon before the conference began, Prof. Niklaus Wirth of Eidgenössische Technische Hochschule in Zurich was awarded an Honorary Professorship at Novosibirsk State University and gave a talk entitled "Threads: Another Paradigm for Multiprocessing".

In general, the high points of the conference were the formal talks and off-podium antics of the European speakers. However, there were strong sessions on Artificial Intelligence, Natural Language Processing, Supercompilation, Partial Evaluation, and Object-Oriented Programming. These topics correspond well with the research areas of the conference organizers. A summary of several exciting talks follows. Revised texts of most talks are available in *Lecture Notes in Computer Science 1181: Perspectives of System Informatics*, Springer-Verlag 1996.

N. Wirth - The Language Lola and Digital Circuit Design

The opening keynote speech, 14:15 Tuesday 25 June, was delivered by Prof. N. Wirth, "The Language Lola and Digital Circuit Design in the CS Curriculum". This excellent presentation discussed the Lola circuit design language, which is used at ETH Zurich to specify wiring of FPGAs, like those from XILINX. To Prof. Wirth's Modula-2 are

added pre-definitions for registers, latches, multiplexors, and gates. There is a replicative FOR statement. Advanced course can define parametrized types, such as counter(N) for all N-bit-wide ripple counters.

Use of Lola allows efficient combination of similar steps in programming and circuit design. Programming has: 1) Algorithm specification in a programming language; and 2) Compilation and execution. Traditional circuit design has: 1) Circuit formulation in HDL (High-level circuit Design Language); 2) Choosing of technology; 3) Placement of components and routing of links; 4) Building the circuit; and 5) Execution and testing. Classroom circuit design with Lola has: 1) Circuit specification in Lola; 2) Layout generation for FPGA using a screen editor; 3) Tests for compatibility of layout and specifications; and 4) Checking of circuit using test programs.

The advantages of Lola for the classroom are: 1) No soldering; 2) Placement and routing through interactive software; 3) Use of the same component (FPGA - field programmable gate array) for all exercises; and 4) Exercises with much more complex circuits. The disadvantage is the difficulty of routing using inexpensive XILINX 3000 cells and connection grid. The 3000 FPGA design is an irregular kludge that is too complex for student use and for practical engineering. Concurrent Logic has a better architecture and is simpler. Lola allows students to specify modular design routing in terms of 4 standard inner cells nested within 4 outer routing patterns.

The XILINX 4000 chip family corrects many of the problems of the 3000 cells. Its cell architecture is better. Many standard cell functions are predefined: 3-input half adders, full adders, latches and latches with feedback for memory. The Lola screen editor lets students select standard cell functions from a menu.

XILINX 4000 FPGAs have 4-direction (NESW) links between cells, allowing regular hierarchical connections. With the newer FPGAs, it takes only a few to route real circuits versus many minutes for the older chips. Lola can do automatic routing of circuits built with regular cells defined for XILINX 4000 FPGA chips.

Since no wire-wrap nor soldering is needed, more complex circuits can be assigned as exercises. A barrel shifter is hard to specify on an FPGA. Most cells are filled with signal links. An 8x8 bit multiplier can be done with 2x3 cell block tiling, if a student is clever. The last class exercise is a UART (Universal Asynchronous Receiver Transmitter) design. One was tested as a real UART chip and was capable of passing 1 Mbit/second.

Advice from the Lola project helped XILINX improve the design of its newer 6200 FPGAs. A course based on Lola is given in the 1995 Springer-Verlag text by Prof. Wirth, *Digital Circuit Design: an Introduction.*

A. Stepanov - Generic Programming

The highlight of the session on Programming Methodology, 16:15-17:00 Tuesday 25 June, was given by Dr. Alexander A. Stepanov of Silicon Graphics in California, "Generic Programming". It was the most interesting talk of the conference.

The dream of this work is generic algorithms that are as efficient as those produced by hand-tailored assembly code, but created rapidly using abstract and efficient software components from a systematically organized library of standard program templates. It has occupied years of work by Dr. Stepanov since his arrival in the United States in the 1970s.

Early studies while at General Electric Research were based on functional languages, which work poorly for graph algorithms that need side effects to store new linkages.

About 1985, while a professor at Brooklyn Polytechnic, he tried ADA and many other procedural languages, but inheritance does not work generically. Dr. Stepanov worked on C++ libraries at Bell Labs and Hewlett-Packard Labs in the late 1980s. In 1993, at HP Labs, his work on generic programming flowered to become the Standard Template Library (STL) for a more modern version of C++. He recently moved to Silicon Graphics.

The approach for generic programming is:

1. Identify useful, efficient algorithms;
2. Make generic representations of these good algorithms;
3. Derive requirements to allow algorithms to run efficiently; and
4. Construct a framework to classify generic algorithms.

The Euclidean algorithm for the greatest common divisor (**gcd**) of two integers **m** and **n**, can be generalized to an entire class of Euclidean elements, e.g., polynomials with integer coefficients. The class of Euclidean elements satisfies the concept of a family of types for which Euclidean **gcd** works. These types have a similar signature, but are not identical. Polynomials can be expressed in terms of their coefficients, which is needed for **gcd** to work, or as the set of their roots, which does not work well for **gcd**.

The simple Dijkstra linear search algorithm in the language C++ is roughly:

```
int *find(int* first, int* last, int value) {
                                /* last is pointer in the last possible matching cell */
while (first != last && *first != value) ++first;  /* *first dereferences first */
return first;}                  /*   if find, return ptr to match; else ptr after the last   */
```

Most sequence operators need an extra value: **n** partitions **=> n+1** boundary points.

The Standard Technical Library (STL) actually has no pointers in its codes. Instead of pointers, the codes use Input Iterators. Each is roughly a pointer to an integer. The core of STL is its axiomatic specifications of iterators, each of which supports operations such as * (dereference), == (test for equality), and ++ advance. Many iterators (I) exist:

1. Input I are unidirectional, read-only, single pass (read once), e.g., find, count, search.
2. Output I allow data storage and are unidirectional, write-only, single pass, e.g., merge.
3. Forward I are unidirectional, multipass (iterating twice gives the same), e.g., rotate.
4. Bi-directional I are multipass, e.g., reverse.
5. Random I give random jumps in constant time.

Iterators define multithreaded algebras, with affiliated types. Iterators effectively generate types.

Most of what is needed for generic programming is in 10% of C++: **while**, **template**, **funct**, **class** without inheritance. The axioms for iterators do not exist in C++. What is not in C++ are iterators, concepts and affiliated classes. Each concept is a family of sets and operators across them.

What needs to be added to C++ to make a Generic C, a C++ for STL?

1. Type constraints need to be explicitly definable.
2. Concepts must become linguistic entities; and
3. Generic functions can have arguments constrained by concept rather than type.

What is needed to transform programming from an art to a science?

1. Concept taxonomy of software components, e.g., matrices; and
2. An easy way to describe both the taxonomy and the invisible concepts.

Most of these good things can be done in C++. There are many good STL tutorials on-line on the web at RPI.edu (Rensselaer Polytechnic Institute), uwisc.edu (U. Wisconsin), Butler.hpl.hp.com (Hewlett-Packard Labs), and soon at SGI.com.

C. Koster - The Making of Algol 68

The first day ended after a17:25-17:50 history lesson by Prof. Cornelis H. A. Koster of the Catholic University of Nijmegen in the Netherlands, "The Making of Algol 68". His droll reporting of human interactions in producing this masterpiece of complexity was as entertaining as his full-volume deep-tenor renditions of classic hymns, arias and Ukrainian love songs after evening meals.

P. Seuren - What a Universal Semantic Interlingua Can Do

The second session on Wednesday 26 June was on Natural Language Processing. It began at 11:00 with an invited talk by Prof. Pieter A. M. Seuren of Nijmegen University in the Netherlands, "What a Universal Semantic Interlingua Can Do". He explained his methods for mapping deep meaning structures revealed by semantic analysis into corrected worded sentences in any of five European languages: English, French, Dutch, German, and Turkish. This mapping is the exact opposite of that done in more common computational linguistics: words imply meaning.

Ideas are expressed as tree structures in a semantic syntax, or Semantic Interlingua, common to those languages. The rules for generating correct English are very simple, only 2.5 pages in length. There are, of course, difficulties with many common verbs in choosing exactly the most equivalent word and getting it in precisely the right place. For example, the German verb *mussen* can be translated into English both as *must* and *have to*, but *must* cannot take auxiliary verbs in English. The German sentence correctly translated as "He has *had to* do it." cannot be expressed as "He has *must* do it.". Careful wording is also needed for verbs in the subjective mood, used to express counterfactual cases in English, German, and French, "If I *were* you, young man, I would work harder.".

This talk covered only half the problems of translating statements in one language into proper statements in another. For automatic translation of German into English, one must perform very careful semantic analysis to get accurate semantic trees expressing the precise meaning.

The proto-Semantic-Interlingua used by Prof. Seuren works so well because it is limited in scope, uses a limited part of the grammar of each language, and targets a limited group of related languages. However, it does provide a simple translation procedure for this important subset of European languages. When it is used, there are very few exceptions to correct by hand.

V. Turchin - Supercompilation: Techniques and Results

The best talks of Thursday, 27 June, occurred in two related sessions on high-level techniques for transforming program sources to make them run faster: Supercompilation and Partial Evaluation. First, at 14:00, was an invited talk by Prof. Valentin F. Turchin, of the City College of New York and formerly of Keldysh Institute in Moscow, "Super-compilation: Techniques and Results". In the late 1960s, he founded the most general of these disciplines, supercompilation. His talk tied his work with that of Futamura (partial evaluation), Ershov (mixed computation), and Jones and Glück (supercompilation).

Prof. Turchin created the functional language Refal, capable of succinctly describing its own semantics, as a vehicle for studying high-level program transformations. The simplest of these are used in Partial Evaluation, automatically to create fast specific programs from generic codes. For example, multiplier code just for 50 x 50 matrices can run twice as fast as the general code for N x N matrices from which it was specialized.

Supercompilation can solve, by general means, many transformation problems in different computing disciplines that usually are considered to be unrelated. He gave many examples. By partial evaluation, a simple algorithm to search a long string for a match to a string pattern can automatically become the maximally efficient Knuth-Morris-Pratt algorithm for matching a specific pattern.

Supercompilation of a function that has nested calls to change all **a**'s in a string to **b**'s, then all **b**'s to **c**'s, and finally all **c**'s to **d**'s in three passes becomes a one pass program that changes all **a**'s to **d**'s, all **b**'s to **d**'s, and all **c**'s to **d**'s. Many other program simplifications were given in the talk.

Prof. Turchin also showed how supercompilation could be used to prove basic theorems: the existence of an additive identity value **0** - for all **x**, **0+x = x**; and that there is no greatest natural number **N**. It can also be used rapidly to solve problems involving inference from existing knowledge: what integer digits substituted for each unknown **?** satisfy the equation 2?5? x 38? = 9??7?3; the only solutions are 2**4**5**1** x 38**3** = 9**38**73**3** and 2**4**5**7** x 38**9** = 9**55**77**3**.

Another application of supercompilation yields a faster solution for the Towers of Hanoi puzzle: given a conical stack of disks on one peg and two empty pegs, move the entire stack to the second peg, making sure that a larger disk never rests upon a smaller one in any stack. The classic recursive solution takes time exponential in **N**, the number of disks in the original stack. The method can automatically be transformed into a fast running solution algorithm linear in **N**.

Prof. Turchin includes a much more complete version of his seminal talk in the final PSI '96 proceedings, published late in 1996 by Springer-Verlag as *Lecture Notes in Computer Science 1181: Perspectives of System Informatics.*

R. Glück - Fast Multi-Level Binding-Time Analysis

The last talk in the Supercompilation session, at 15:10, was also interesting. Prof. Robert Glück of the University of Copenhagen, Denmark, spoke on "Fast Multi-Level Binding-Time Analysis (BTA) for Multiple Program Specialization." He gave an excellent overview of the use of partial evaluation for automatic generation of compilers and parsers from interpreters. His work extends early work by Turchen. It uses multi-level meta-evaluation of programs to transform them much faster, in nearly linear time, than possible with uni-level supercompilation.

Glenstrup - BTA Algorithms to Ensure Partial Evaluation Termination

The following session on Partial Evaluation also contained two interesting papers about program transformation techniques. The first, at 16:00, was delivered excellently by a young third-generation researcher in the heretofore esoteric field of program transformations. Arne Glenstrup, working with Prof. Neil Jones at the University of Copenhagen (in turn inspired by Profs. Futamura, Turchen, and Ershov), spoke on "BTA Algorithms to Ensure Termination of Off-Line Partial Evaluation".

Program specialization systems currently have many failings. They may loop infinitely when given certain programs and static parameter values; they may give enormous, impractical transformed programs; or they may produce programs that run more slowly than the originals. Most systems require on-line human tuning or off-line program annotations to direct the specializer to try certain transformations but not others.

Jones and Glenstrup have developed binding time analysis (BTA) algorithms that let specializers handle tail recursion in programs without looping, thereby guaranteeing termination. The BTA algorithms are able to find all portions of a program that will be static if some parameter value is static. They automatically generate annotations to tell off-line specializers every part of a program to check to see if it can be simplified, and which parts to ignore because they are dynamic or must be treated as dynamic ("lifted") to guarantee specializer termination. Their work is a promising step on a path to create practical systems to produce fast versions of generic algorithms for specific applications.

P. Thiemann - Polyvariant Expansion and Compiler Generators

The next paper in the same strong session was delivered, at 16.25, by Peter Thiemann of the University of Tübingen, Germany, "Polyvariant Expansion and Compiler Generators". He shows how to use partial evaluation based on simple monovariant binding time analysis (BTA) to duplicate parts of a program systematically to allow more precise determinations of binding times and generate faster running code. Their method gives the program transformation precision of polyvariant BTA at the greatly reduced cost of monvariant analysis. Polyvariant expansion is similar in spirit to variable renaming in optimizing compilers, but whole program functions are renamed and duplicated, not just single variable names that have very different meanings in various parts of a program.

Pree - Framework Component Systems

The most interesting paper of the final day, Friday, 28 June, was delivered at 9:45 in the session on Object-Oriented Programming. The speaker was Wolfgang Pree of the University of Linz, Austria. The talk was entitled "Framework Component Systems: Concepts, Design Heuristics, and Perspectives". Frameworks are semifinished program architectures, which are systems of preplugged code modules that often are much more reusable in practice than isolated modules. The paper gives heuristics for determining where software system designs will often need flexibility, so-called domain-specific hot spots. It shows how to develop software frameworks with components prespecified at all but the most likely hot spots. If correctly designed, the resulting software can be re-used for many applications with minimal effort. Code that rarely needs changing is always present. Only critical design points must be elaborated for each application.

Conclusions About PSI '96

There were strong sessions on artificial intelligence, program transformations, and object-oriented programming. Many other topics in parallel computing and systems programming were covered. The program transformation papers represented the work of the key practitioners in the world in an esoteric, but increasingly important field. This conference cast its net over the many areas of computer science of interest to the

late Prof. Ershov, founder of the sponsoring Institute of Informatics Systems. Despite its breadth, it was a good conference, full of intelligent, entertaining participants.

2. TRIP TO PERESLAVL-ZALESSKY JULY 1996

After leaving Novosibirsk, the investigator spent 12 days, 29 June - 11 July 1996, working with colleagues at the Keldysh Institute of Applied Mathematics in Moscow. The high point of this working visit was a three-day trip to participate in a seminar on supercompilation given one week every summer by Prof. Valentin Turchen.

The annual seminar allows Prof. Turchen to meet his ex-students and their students. It is held in Pereslavl-Zalessky, a beautiful 852-year-old fortress and monastery town of 100,000 people about 200 kilometers north of Moscow. People in the ancient fortress center dwell in hundreds of classic Russian wooden houses. They contrast greatly with the concrete apartment towers constructed in Russian cities this century.

The investigator was struck by the almost universal applicability of Prof. Turchen's supercompilation techniques during his invited lecture at PSI '96. One of his colleagues at Keldysh studied under Prof. Turchen in Moscow before Prof. Turchen was forced into exile in the mid-1980s because of his support of Sharansky and the Soviet human rights movement. The colleague was able to get permission and automobile rides for the investigator to attend the first part of the seminar. Prof. Turchen is a very special human being.

While in Pereslavl-Zalessky, the investigator was introduced to another computer scientist who studied under Prof. Turchen: Dr. Sergei Abramov. Prof. Abramov is director of the Research Center for Multiprocessor Systems (RCMS), a very active part of the Program Systems Institute of the Russian Academy of Sciences located in Pereslavl. Prof. Abramov is very energetic and barely in his forties, young for a Center director in Russia. He is also founder of the University of Pereslavl, located just outside the grass- and goat-covered fortress wall in the heart of the ancient town, a place of lush beauty.

Prof. Abramov heads 25 researchers in computer science and mathematics. RCMS consists of three laboratories: Program Systems for Parallel Architectures, Programming Automatization, and Problems of Small Dimension. The main work is on multiprocessor systems software, including systems to parallelize applications automatically and dynamically during execution and to reconfigure parallel software to accommodate changes in hardware availability and to tolerate faults. A specific application of this research is software being developed to run commercially important parallel programs over processors in fast TCP-based networks. The adaptive operating system for parallel processing on networks is being implemented for collections of both Intel-based PCs and Inmos-based multiprocessors.

Other efforts by RCMS personnel includes management of the "botik.ru" (sail boat - Alexander Nevski built the first Russian Navy in nearby lake waters) regional hub in the growing Russian internet, work with supercompilation and functional programming languages, and mathematical studies of various algebraic and topological systems. Besides maintaining a pool of telephone modems for network access, RCMS runs a sophisticated Web site at www.botik.ru, has strung kilometers of fiber-optics cable on above-ground poles as a regional high-bandwidth data backbone, and has developed inexpensive but very fast PC-based gateways connecting network segments. Two

graduate researchers, Andrei Nemytykh and Victoria Pinchuk, working directly with Prof. Abramov presented a paper in the Supercompilation session at PSI '96 in Novosibirsk. Two other Program Systems Institute researchers from Pereslavl also attended the conference.

3. CONCLUSIONS.

The Investigator made eight trips to Russia between March 1994 and August 1996, seeking opportunities for joint Russian American research in computer science, and especially in the fields related to high performance computing software and hardware. Of all these trips, the most intellectually stimulating was this seventh one to PSI '96 in June 1996 and to Moscow and Pereslavl-Zilessky immediately afterwards.

The Investigator met many very bright computer scientists and mathematicians in Russia. However, too many in Moscow seem preoccupied with power and money, politics and commerce. Those in Novosibirsk seemed even brighter on average, but somewhat reticent, at least with visiting American strangers. They also seemed slightly more stunned by recent economic changes in Russia and the new hardships for life in Siberia. It was only in Pereslavl-Zalessky that people were openly and actively getting on with their lives. They were poor by western monetary standards, but full of warmth, joy and dreams. History and beauty fills their region. They are rich in human terms.